CHAPTER 7

STORING AND RETRIEVING PROGRAMS AND DATA

Up to this point, we have dealt with reading data, using READ and DATA statements, inputting data, using the INPUT statement, and printing output. We have not discussed storing and retrieving programs or storing and retrieving data. It seems useless to write a program, key it in, get it to work, and then be unable to save it to run again later without keying it in again. As you write programs to solve more complex problems with large amounts of data, you'll need ways to save your programs and data on some type of storage medium. You will probably use a magnetic disk or tape.

The way this is done varies considerably from one computer to another. Therefore, we will not attempt to present all the possible ways. The primary concern is that you are aware that there are ways to store and retrieve both programs and data. Your computer user's reference manual will provide you with detailed instructions.

STORING AND RETRIEVING PROGRAMS

First, let's examine one method for storing a program. When storing a program, you must have a way to reference it in order to load it to be run again. To do this, you assign it a name. Program names are a series of characters you choose. Depending on the computer you are using, you will probably be limited to six or eight characters. There may also be limitations on specific characters that can be used. Most computers require the first character in a program name to be alphabetic.

If you are using a computer which has more than one programming language available, you will have to specify the programming language you wish to use. The following example shows how this might be done.

Example:

```
SYSTEM          BASIC
OLD OR NEW      NEW
NEW FILE NAME   MPG
READY
```

In the first line of the example, the computer asks what programming language you want to use by displaying the word, SYSTEM. You enter BASIC; then it asks if you want to retrieve an OLD program or enter a NEW one. You enter NEW. The computer then asks for the NEW FILE NAME; you enter MPG as the program name. The computer then responds with the message, READY. This indicates the computer is ready for you to key in your program. You key in your program in the same manner as you have throughout this course.

```
10    LET M1 = 0
20    LET G1 = 0
30    DATA 485,16.5,450,14,418,11,432,12.5
40    FOR X = 1 TO 4
50    READ M,G
60    LET M1 = M1 + M
70    LET G1 = G1 + G
80    NEXT X
90    LET A = M1/G1
100   PRINT ''TOTAL MILES ARE'';M1;''TOTAL GALLONS ARE'';G1
110   PRINT ''AVERAGE MILES PER GALLON IS'';A
999   END
SAVE
FILE MPG SAVED
```

Once you have entered the program, you key in the system command, **SAVE,** which instructs the computer to store your program either on disk or tape. The computer then responds with the message, FILE MPG SAVED.

To recall a saved program, you respond with OLD to the computer query NEW or OLD. The computer will then ask for the program name. After you have entered the program name, the computer loads the program into memory. It is then available to be executed, changed, or listed.

To SAVE a program on some computers, all that is required is to key in the program, enter the system command, SAVE, followed by the program name enclosed in quotation marks.

Example:

```
SAVE ''MCASH''
```

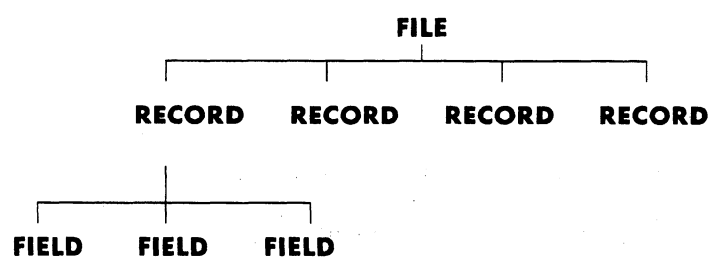
This command will save the program.

STORING AND RETRIEVING DATA

The programs we have written thus far dealt with data either stored in the program with DATA statements or input via an INPUT statement at time of program execution. These statements may be sufficient when handling data that is to be processed only once. However, there are times when you will want to save your data to be used by other programs. You are restricted in what you can do with the data when it is stored in DATA statements in a program. It is impractical to update data or to use the same data with different programs. It would be very time consuming to go through all the DATA statements and make changes, or to enter all new DATA statements each time you wanted to update and/or process the data. To eliminate having to enter the data each time it is to be processed, it can be stored independently on an auxiliary storage medium such as disk or tape. This keeps the data separate from the program so that it can be used with different programs without being rekeyed. Data stored in this manner is called a data file.

A *data file is* any group of related records, such as inventory, personnel, payroll, manhour accounting, and so on. A *record* is composed of *data fields* which are specified areas of a record used for a particular category of data. For example, in a parts inventory file you would have a record for each part stocked. Each part and all the associated data about that part (part number, part name, and unit price) would make up a record. Part number, part name, and unit price are each a field. The following example shows a pictorial breakdown of a file composed of records and associated fields.

Example:



You might say, "All of this is fine, but how do I create and use these files?" Since the BASIC file processing statements are not standardized, you will have to refer to the user's reference manual for your computer to find the file processing statements and their syntax. We will attempt only to deal with the file processing statements conceptually.

Solving Problems With File Processing Statements

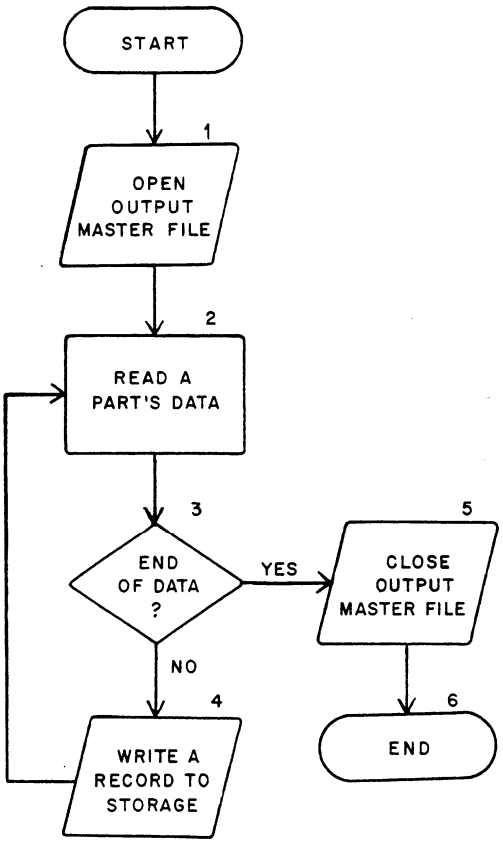
Suppose you were keeping track of an inventory file for an auto parts store. You would need to create a file which would contain a record for each part in the inventory. This type of file is called a master file. Each record will contain the part number, part name, on-hand quantity, unit price, and reorder

point (stock level). The flowchart in figure 7-1 describes one procedure that could be used to create the master file.

Before you can write data records to a storage medium, you usually have to set up the file. Most computers do this with some form of *open* statement that names the file and makes it available for processing (block 1). Once the file is available for processing, the program will read the data elements from DATA statements just as in the past (block 2). Then the program checks for end-of-data (block 3); if there is data, it will write a record to the output file for each part in the inventory (block 4). Once all the data has been read and written you must tell the computer you are finished with that file. This is usually done with some form of close statement (block 5). At this point the computer does any necessary end-of-file processing then terminates the program (block 6).

With a fundamental understanding of the problem and a flowchart as a guide, we are ready to write a program to create the master file: The program could be written as shown in figure 7-1.

The **DATA SAVE** statement in line 10 opens the output file, making it available for the program to write data out to it, and assigns it the file name, INVFILE. The #2 tells the computer what storage medium is being used. Lines




```
05   REM THIS PROGRAM CREATES A MASTER FILE
10   DATA SAVE #2,OPEN ''INVFILE''
20   READ P,N$,Q,U,R
30   IF N$ =''ENDDATA'' THEN 60
40   DATA SAVE #2,P,N$,Q,U,R
50   GOTO 20
60   DATA SAVE #2,END
200  DATA 0517,''SOCKET'',8,2.95,5
210  DATA 7340,''SHACKLE'',10,12.48,10
220  DATA 1034,''SPARKPLUGS'',25,16.42,15
230  DATA 3054,''TIRES'',9,64.50,4
240  DATA 9121,''BRAKE SHOES'',20,15.45,26
250  DATA 5230,''BATTERY'',16,49.00,8
260  DATA 6120,''POINTS'',50,6.40,20
270  DATA 5510,''CONDENSER'',12,3.62,20
500  DATA 0,''ENDDATA'',0,0,0
999  END
```

Figure 7-1.—Flowchart and program to create a master file.

20 through 50 form a loop which reads data from DATA statements and writes records to the master file until all the data has been read and written. In line 20, the READ statement reads the data from the DATA statements, and line 30 tests the data for the "ENDDATA" record which indicates end-of-data. If it is not end-of-data, then the DATA SAVE statement in line 40 writes a record containing the values for the variable names listed. It writes the record onto the master file (INVFILE). The GOTO statement in line 50 transfers control back to line 20 to repeat the reading, testing, and writing of records until the READ statement reads the "ENDDATA" record and the IF-THEN test proves true. Then the program transfers control to line 60 which writes an end-of-file character on the master file and closes the file. The next executable statement is 999 which terminates execution of the program.

For simplicity we have included only the data for one record in each DATA statement. This makes it easier to visualize what a record looks like. It also makes it easier to correct any data that is keyed incorrectly. Notice also that the DATA statements begin with much higher line numbers. This separates them from the program steps and allows room for expanding the program or for the addition of data without having to renumber the program. Using "ENDDATA" to indicate end-of-data rather than using a FOR-NEXT loop allows more flexibility in handling various amounts of data and eliminates having to know how much data will be processed.

In this example, the DATA SAVE statements are used to open a file and write a file name on it (line 10), write data to the output file (line 40), and write an end-of-file indicator on the file (line 60). Since the file processing statements will probably be different on the system you are using, you will have to refer to the user's reference manual to determine what statements perform these functions.

Once the master file has been created you might want to print it, either to ensure that all items have been entered and entered correctly; or you may wish to have a listing on which you can keep a tally of items sold and use it when you update the master file. The following program could be used to print the master file.

Example:

```
10  REM THIS PROGRAM PRINTS MASTER FILE INVFILE
20  DATA LOAD #2,''INVFILE''
30  PRINT ''PART NUMBER'',''PART NAME'',''QUANTITY'',''PRICE'',
40  PRINT ''REORDER POINT''
50  DATA LOAD #2,P,N$,Q,U,R
60  IF END THEN 99
70  PRINT P,N$,Q,U,R
80  GOTO 50
99  END
```

In this program, the **DATA LOAD** statement, line 20, opens the master file (INVFILE) and makes it available for processing. The PRINT statements in lines 30 and 40 print the column headings. Lines 50 through 80 comprise the loop which will control the reading and printing of the master file. The DATA LOAD statement in line 50 reads the values for the variables and makes them available for the PRINT statement to print. The **IF END** statement in line 60 checks for end-of-file. At end-of-file, control is transferred to line 99 where the END statement terminates the program.

When reading a file and the end-of-file indicator is read, the computer will detect it. If any future attempt is made to read from the file before it is reopened, an end-of-file error message is printed and the computer stops executing the program. When reading a file, the BASIC statement IF END can be used to test whether the computer did, in fact, read the end-of-file indicator. It also transfers control to any specified line number in a program. For example, you could transfer control to a PRINT statement to print summary totals, or transfer control to an END statement to terminate the program as was done in this example. Depending on what computer you are using, the IF END statement is normally placed either immediately following or immediately preceding a read statement.

Now that we have created and printed the master file, we can now begin to use it. For example, we'll need to update the file so the on-hand quantity for each item will be current and we'll know when a particular item needs to be reordered. The flowchart (figure 7-2, a foldout at the end of this chapter) is a pictorial representation of the procedures required to solve the problem. The problem solution has been divided into four parts:

1. Read the part's data into arrays.
2. Update the quantity on hand.
3. Produce a list of items below stock level.
4. Create a new master file.

While some of these parts could have been combined, they were kept separate for ease of understanding.

Study the flowchart and examine its four distinct parts. Part one consists of blocks 1 through 4 which are used to read the parts data into arrays and to accumulate a total count of the records in the file. This total will be used to control successive FOR-NEXT loops.

Part two consists of blocks 5 through 12 which are the update portion. This section allows you to select part numbers at random to be updated. Using the part number entered in block 5, the FOR-NEXT loop compares it to each part number in array P. When a match is found, the subscript X equals the part's relative position in the arrays and control is transferred to block 11. Block 11 allows you to enter quantity sold. Block 12 uses the value of X as the subscript to select the corresponding on-hand quantity, Q(X), to be updated. Q(X) is used with the quantity sold, Q1, to compute the new on-hand quantity. Then control is transferred back to block 5 for another part number to be entered. If a match is not found, this means the entire array has been searched in the FOR-NEXT loop, blocks 7 through 9, and the part number (P 1) entered in block 5 did not match any part number in the parts number array, P(X). Therefore, an error message is printed and control is transferred back to block 5 to enter the corrected part number

or another part number. If no more updating is required, 9999 is entered and block 6 tests this condition. If it proves true, control is transferred to block 13 which is the first block of part three.

Part three consists of blocks 13 through 17 which generate a list of parts below stock level. Blocks 14 through 17 comprise a FOR-NEXT loop. Block 15 tests if the on-hand quantity is greater than the reorder-point quantity. If this proves false, the part number, part name, and unit price of that part is printed on the below stock level list. Block 17 tests for X equal to K. If it is not equal, control is transferred to block 14 which increments the value of X by one and continues searching the records for items to be reordered. Once X is equal to K (the number of records in the file), control is transferred to block 18, which is the first block of part four.

Part four consists of blocks 18 through 22 which create the new master file. Blocks 19 through 21 comprise a FOR-NEXT loop which is used to control the writing of records to the new master file. Block 20 writes the new records onto an auxiliary storage medium; block 21 tests for X equal K. If X-is not equal to K, control is transferred back to block 19 which increments X by 1 and block 20 writes another master record. Once X is equal to K, control is transferred to block 22 which closes the new master file and the program is terminated, block 23.

Examine the program (figure 7-2) and see how the program coding corresponds to the blocks in the flowchart.

In this program, lines 50 through 130 are used to read the master file into the computer's memory. Lines 150 through 300 are used to update the master file. In line 190 the part number to be updated is entered; or if no more parts are to be updated, 9999 is entered. Line 200 tests P1 for end of update data (9999); if not, control goes to line 210 which is the beginning of a FOR-NEXT loop that searches array P for a matching part number. If a matching part number is not found, control is then transferred to line 240 which prints the message, PART # NOT FOUND. If a matching part number is found, control is transferred to line 260 which displays the data elements of the record to be updated. Line 270 requests quantity sold to be entered. Line 290 computes new on-hand quantity by subtracting Q1 from Q(X). Then the GOTO statement in line 300 returns control to line 180 which is a prompt to enter another part number to be updated.

When no more parts records are to be updated, control is transferred to line 340 which prints the heading for the below stock level list. Lines 350 through 380 comprise a FOR-NEXT loop which is used to search the on-hand quantity and the reorder point quantity arrays. Line 360 tests the updated on-hand quantity, Q(X), and the reorder point quantity, R(X), to determine if Q(X) is greater than R(X). If it is, the data for another record is tested. If it is less than or equal to R(X), the part number, part name, and unit price are printed on the list. Once all parts have been tested for reorder criteria, lines 400 through 470 write the updated master file to an auxiliary storage medium.

In this program we chose to read all the data into arrays to allow updating the parts records in any sequence, and to allow a part's record to be updated more than once in a single run. This will work for small data files. However, for larger data files (those that cannot fit into memory) other processing methods would be needed.

SUMMARY

As you write programs to be used repetitively and to solve more complex problems, the need for storing and retrieving programs becomes more apparent. Although the specific statements for storing and retrieving programs will vary on different computers, the concepts will be similar. Programs must be assigned a name so they can be referenced and reloaded. A program name is a series of characters used to identify a program. A system command, such as SAVE, is used to store a program. To reload a program will require another system command in which you specify the name of the program to be loaded. Once a program is loaded into a computer's memory, it is available to be executed, changed, or listed.

The need for a method to store and retrieve data files becomes apparent when the same files are to be used extensively. When data is stored in a program, you are restricted in what can be done with it. This data cannot be used with other programs and it would be very difficult, if not impossible, to update. To eliminate these restrictions, data can be stored on auxiliary storage media such as magnetic disk or tape. Data stored in this manner is called a data file. A file consists of a group of related records and each record consists of fields. A field is a specified area in a record used for a particular category of data.

Most computers require a file processing statement to make a file ready to be read or written. This is usually some type of *open* statement and includes the file name and the tape or disk where the data will be written or read. Once the file is opened, file processing statements are used to read data from a storage medium into memory or to write data from memory to a storage medium. Once you are through processing a data file, you must tell the computer you are finished with that file. This is done with some type of close statement.

Storing data independently from programs allows more flexibility. Data stored in this manner can be used by different programs. It can be updated, printed, or used to generate various reports.

The programming concepts presented in this text are fundamental; they are intended to get you started in programming. As your knowledge and experience increase, you will discover new and more sophisticated ways to use the language to solve more difficult problems.

CHAPTER 7

EXERCISES

1. Draw a flowchart and write a program to create a file to keep track of stock purchases. Use the file processing statements presented in this chapter. The file should include a record for each stock purchased. Each record should be composed of three data fields: name of stock, purchase price per share, and number of shares bought. Use the INPUT statement to enter the data for the file.

2. Four candidates are running for public office. To win, a candidate must receive over 50% of the votes cast. Draw a flowchart and write a program to accumulate total votes for each candidate and determine if a candidate has won or if a runoff election is required. Print the result and print each candidate's name and total votes for each. The file to be used as input is stored on tape or disk and consists of records from six districts. Each record contains the following fields:

   1. Candidate number (1-4)—variable X

   2. Candidate name—string variable N$

   3. District number (1-6)—variable D

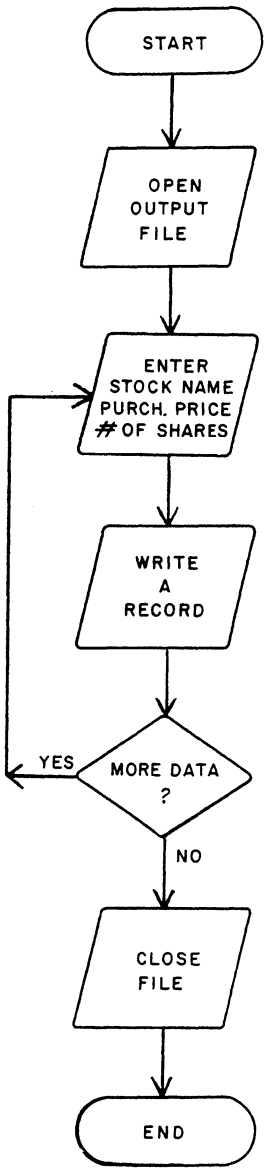
   4. Number of votes—variable V

   The records in the file are in no particular order.

CHAPTER 7
EXERCISE SOLUTIONS

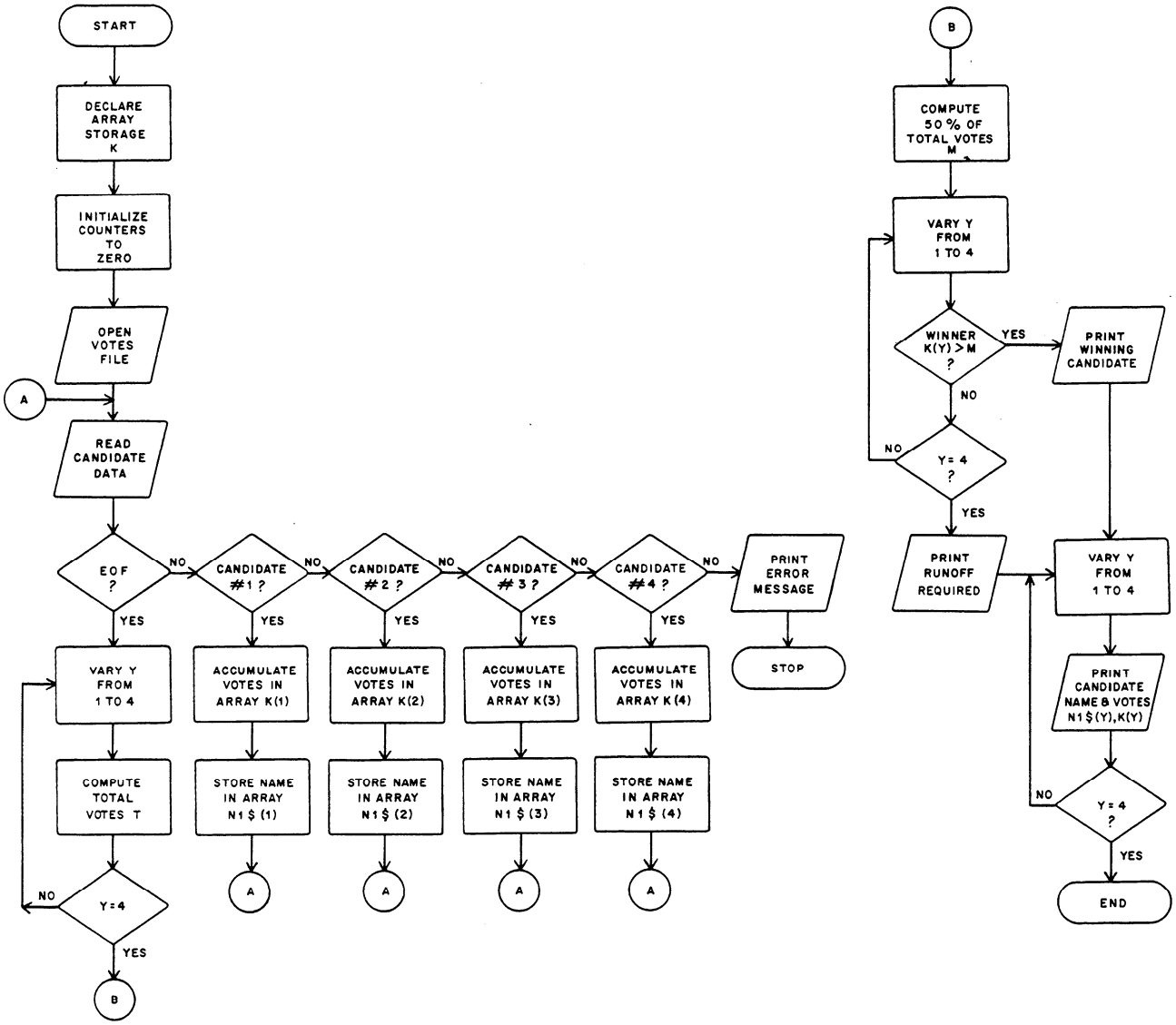
The following flowcharts and programs present possible solutions to the exercises.

1. Stock Purchase Flowchart and Program




```
05  REM STOCK PURCHASE PROGRAM
10  DATA SAVE #2,OPEN"STKFILE"
20  PRINT "ENTER STOCK NAME,PURCHASE PRICE,# OF SHARES"
30  PRINT "ENTER END,0,0 IF NO MORE DATA"
40  INPUT N$,P,S
50  IF N$ = "END" THEN 80
60  DATA SAVE #2,N$,P,S
70  GOTO 40
80  DATA SAVE #2,END
99  END
```

2. Election Results Flowchart and Program



```
10   DIM K(4)
15   DIM N1$(4)
20   LET T = 0
30   FOR Y = 1 TO 4
40   LET K(Y) = 0
50   NEXT Y
60   DATA LOAD #2,"VOTES"
70   DATA LOAD #2,X,N$,D,V
80   IF END THEN 230
90   ON X GOTO 110,140,170,200
100  PRINT "ERROR IN CANDIDATE
105  STOP
110  LET K(1) = K(1) + V
120  LET N1$(1) = N$
130  GOTO 70
140  LET K(2) = K(2) + V
150  LET N1$(2) = N$
160  GOTO 70
170  LET K(3) = K(3) + V
180  LET N1$(3) = N$
190  GOTO 70
200  LET K(4) = K(4) + V
210  LET N1$(4) = N$
220  GOTO 70
230  FOR Y = 1 TO 4
240  LET T = T + K(Y)
250  NEXT Y
260  LET M = .5*T
270  FOR Y = 1 TO 4
280  IF K(Y) > M THEN 320
290  NEXT Y
300  PRINT "RUNOFF REQUIRED"
310  GOTO 330
320  PRINT N1$(Y),"WON!"
330  FOR Y = 1 TO 4
340  PRINT N1$(Y),K(Y)
350  NEXT Y
999  END
```

You may have noticed that the data field for candidate number contains a 1, 2, 3, 4, and that this code could be used as a subscript to reference the correct locations in the arrays. To modify the program to take advantage of this, you could replace statements 90 through 220 with the following statements:

```
90   LET K(X) = K(X) + V

100 LET N1$(X) = N$

110 GOTO 70
```

When the value of candidate number is 1, the number of votes cast for candidate 1 would be added to K(l); when it's 2, the number of votes cast for candidate 2 would be added to K(2); and so on.

When you are defining data and data codes, it is a good idea to think about how you are going to use the data in the programs. You maybe able to save yourself coding time.

THIS PAGE INTENTIONALLY LEFT BLANK.

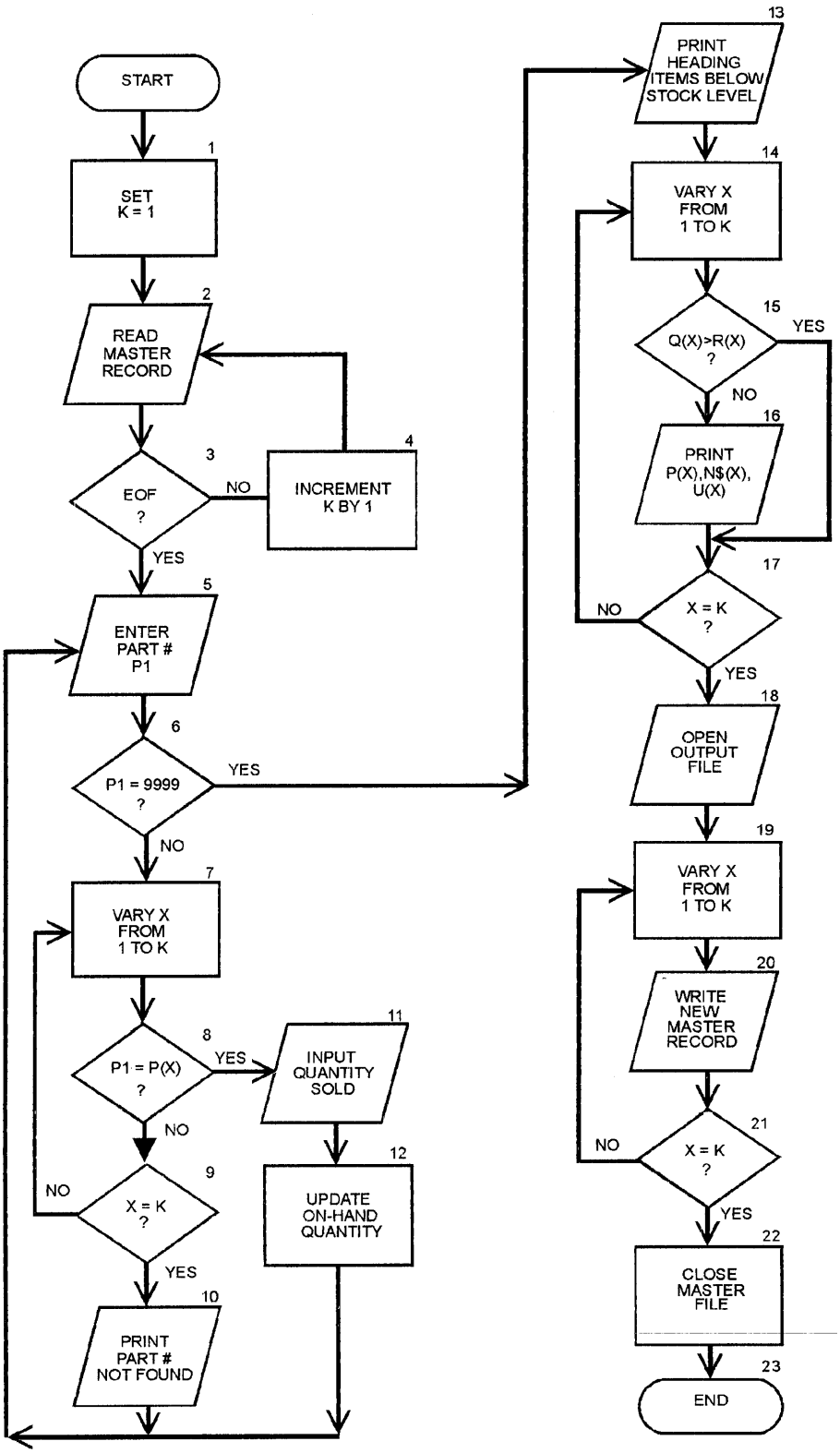


Figure 7-2.—Parts inventory flowchart and program.

```
10      REM THIS PROGRAM UPDATES THE PARTS MASTER FILE,
20      REM PRINTS A LIST OF PARTS BELOW STOCK LEVEL AND
30      REM CREATES A NEW MASTER FILE
40      REM
50      REM PART 1 (BLOCKS 1-4) LOAD THE MASTER FILE
60      REM
70      LET K = 1
80      DIM P(8),N$ (8),Q(8),U(8),R(8)
90      DATA LOAD #2,"INVFILE"
100     DATA LOAD #2,P(K),N$(K),Q(K),U(K),R(K)
110     IF END THEN 170
120     LET K = K+1
130     GOTO 100
140     REM
150     REM PART 2 (BLOCKS 5-12)  UPDATE THE ON-HAND QUANTITY
160     REM
170     LET K = K - 1
180     PRINT "ENTER PART #,9999 IF NO MORE DATA"
190     INPUT P1
200     IF P1 = 9999 THEN 340
210     FOR X = 1 TO K
220     IF P1 = P(X) THEN 260
230     NEXT X
240     PRINT "PART # NOT FOUND"
250     GOTO 180
260     PRINT P(X),N$(X),Q(X),U(X),R(X)
270     PRINT "ENTER QUANTITY SOLD"
280     INPUT Q1
290     LET Q(X) = Q(X) - Q1
300     GOTO 180
310     REM
320     REM PART 3 (BLOCKS 13-17) PRINT THE BELOW STOCK LIST
330     REM
340     PRINT "THESE ITEMS ARE BELOW STOCK LEVEL"
350     FOR X = 1 TO K
360     IF Q(X)>R(X) THEN 380
370     PRINT P(X),N$(X),U(X)
380     NEXT X
390     REM
400     REM PART 4 (BLOCKS 18-21) WRITE THE UPDATED
410     REM MASTER FILE TO STORAGE
420     REM
430     DATA SAVE #3,OPEN "INVFILE"
440     FOR X = 1 TO K
450     DATA SAVE #3,P(X),N$(X),Q(X),U(X),R(X)
460     NEXT X
470     DATA SAVE #3,END
999     END
```

Figure 7-2 cont'd.—Parts inventory flowchart and program.